coffea docs

Release 0.4.0

-

March 17, 2015

Contents

event based and extensible irc client library with multi-network support

For support, report an issue on github or join our IRC channel at

CHAPTER 1

quickstart

This is all the code needed to get you started with coffea!

```
var client = require('coffea')(['chat.freenode.net', 'irc.oftc.net']); // or put just one network as

client.on('motd', function (err, event) {
    client.join(['#foo', '#bar', '#baz'], event.network);
});

client.on('message', function (err, event) {
    console.log('[' + event.network + '][' + event.channel.getName() + '] ' + event.user.getNick() +
    //[freenode][#foo] nick: message
    event.reply(event.message); // I'm a parrot
});
```

1.1 with ssl

If you want to enable SSL for a connection, you have to use a network config object when connecting:

```
var client = require('coffea')({
    host: 'chat.freenode.net',
    ssl: true
});
```

CHAPTER 2

api

The best way to get started with coffea is playing around with it! You can use the *genindex* to get a list of `functions` and `events` you can make use of.

coffea function: A function called with `client.function()` if coffea was imported to `client` (see Quickstart)

```
client.function();
```

coffea event: You can listen to an event by defining an event listener like this (make sure to replace `event` with the event name):

```
client.on('event', function (err, event) {
    if (err) throw err; // something bad happened!

    console.log(event); // do something with event here
});
```

Go to API reference or browse the API by...

2.1 plugins

2.1.1 connection api

class `Client` (*info*)

Arguments

- **info** (*object*) – details for IRC connection

Initialise a server connection using the `info` parameter. `info` can be a javascript object representing server details or an array of multiple server details.

Example:

```
var client = Client({
    'name': 'freenode', // Optional
    'host': 'chat.freenode.net', // Required
    'port': 6697, // Optional
    'ssl': true, // Optional
    'nick': 'nickname', // Required
    'username': 'test', // Optional
    'realname': 'Testing Robot' // Optional
});
```

add(*info*)

Arguments

- **info** (*object*) – Details for new connection.

Returns The stream_id.

Add another network to an existing client, using the `info` parameter (as described at `Client()`), since the API supports multiple servers, more than one `add` calls can be specified throughout your program.

Multi-server example:

```
client.add([{ // existing 'Client' object
    host: 'chat.freenode.net', // required
    name: 'freenode',  // optional, but aids in identification, when referenced later in the program
    nick: 'test', // required
    ssl: true, // optional
    username: 'test', // optional
    'realname': 'Testing Robot' // optional
    },
    {
    host: 'chat.freenode.net',
    name: 'freenode',
    nick: 'test2',
    ssl: false,
    username: 'test',
    'realname': 'Testing Robot'
    }
]);
```

Single-server example:

```
client.add({ // existing 'Client' object
  host: 'chat.freenode.net', // required
  name: 'freenode',  // optional, but aids in identification, when referenced later in the program.
  nick: 'test', // required
  ssl: true, // optional
  username: 'test', // optional
  'realname': 'Testing Robot' // optional'
  })
```

write(*str*, *network*, *fn*)

Arguments

- **str** (*string*) – The string to be written.
- **network** (*object*) – The network that the string shall be written to.
- **fn** (*function*) – The callback function to be called when the `write` call has been finished.

Returns string stream_id The stream ID from the call.

Writes raw data (`str`), to `network`, when finished, calls `fn`.

2.1.2 event api

Events are applied using the following example (basically standard NodeJS EventEmitter syntax):

```
client.on('EVENT', // 'EVENT' being any of the events listed in the documentation.
  function(err, event) { // callback function, called when event is fired
        if (err) console.error('Event error:', err);
    console.log('Something happened!');
  });
```

event object

You may have noticed that we are passing an event argument to the event binding function. This is actually not just data but an object with an API to help you deal with events without hassle. Not all events are the same, so please check the various plugin documentations to see what events are available. The following functions and attributes are always available:

network

The network this event was triggered in.

Note: The reply functions are only going to succeed when the `channel` or `user` attribute is available.

---

reply

Answer to a message (same channel/query as the event came from).

replyAction

Answer to a message with an action (`/me`).

replyNotice

Answer to a message with a notice.

core events

ssl-error

The `ssl-error` event, fired when there was an error establishing an SSL connection. If you're running with `ssl_allow_invalid` this event will still fire, but coffea will continue connecting to the server afterwards.

Event attributes:

- None

Example:

```
client.on('ssl-error', function (err, event) {
        console.error('SSL Error:', err);
});
```

disconnect

The `disconnect` event, fired when the client was disconnected from a network.

Event attributes:

- None

Example:

```
client.on('disconnect', function (err, event) {
        console.log("We disconnected!");
});
```

2.1.3 user api

class `User` (*nick*, *client*, *network*)

Arguments

- **nick** (*string*) – The users nickname
- **client** (*object*) – Client object.
- **network** (*object*) – Network object.

Create a new user object.

`User.toString()`

Returns string hostmask Hostmask of the user.

`User.getNick()`

Returns string Nick Nick of the user.

`User.getNetwork()`

Returns string Network Network of the user.

`User.getUsername()`

Returns string Username Username of the user.

`User.getRealname()`

Returns string Realname Realname of the user.

`User.getHostname()`

Returns string Hostname Hostname of the user.

`User.getAccountName()`

Returns string AccountName AccountName of the user.

`User.getChannels()`

Returns object Channels Channels of the user. `{'#channel': ['~']}`

`User.getServer()`

Returns string Server Server the user is connected to.

`User.getServerInfo()`

Returns string ServerInfo ServerInfo of the Server the user is connected to.

`User.getAway()`

Returns boolean Away Away status of the user.

`User.getAccount()`

Returns string Account Account of the user.

`User.isRegistered()`

Returns boolean registered Registration status of the user.

`User.isUsingSecureConnection()`

Returns boolean secureConnection SSL status of the user (on/off).

`User.getIdle()`

Returns int Idle Idletime of the user.

`User.`**`getSignonTime`**`()`

Returns string SignonTime SignonTime of the user.

`User.`**`isOper`**`()`

Returns boolean oper Oper status of the user.

`User.`**`notice`**`(`*msg*`)`

Arguments

- **msg** (*string*) – Notice message to send to the user.

`User.`**`say`**`(`*msg*`)`

Arguments

- **msg** (*string*) – Message to send to the user.

`User.`**`whois`**`(`*fn*`)`

Arguments

- **fn** (*function*) – The callback function to be called when the call has been finished.

events

nick

The `nick` event, fired when someone changes nickname.

Event attributes:

- `user` - User who changed nick.
- `oldNick` - Their old nickname.

Example:

```
            client.on('nick', function (err, event) {
                console.log(oldNick + " is now " + event.user.getNick());
});
```

whois

The `whois` event, fired when the whois response is received from the server.

Event attributes:

- `whoismap` - `{user: whois_data}`.

Example:

```
            client.on('whois', function (err, event) {
    if (err) console.err("Couldn't whois:", err);
    console.log(event);
});
```

functions

getUser (*nick*, *network*)

Arguments

- **nick** (*string*) – The user you want to get by nickname.
- **network** (*object*) – The network to execute the command on.

Get a user object by nickname.

isUser (*user*)

Arguments

- **user** (*object*) – The user object you want to check.

Checks if the passed object is a valid user object.

whois (*target*, *network*, *fn*)

Arguments

- **target** (*object*) – The `channel` or `user` object to send this message to.
- **network** (*object*) – The network to execute the command on.
- **fn** (*function*) – The callback function to be called when the call has been finished.

Send a whois request to the server.

identify (*username*, *password*, *network*, *fn*)

Arguments

- **username** (*string*) – The username to identify with.
- **password** (*string*) – The password to identify with.
- **network** (*object*) – The network to execute the command on.
- **fn** (*function*) – The callback function to be called when the call has been finished.

Identifies the user with nickserv.

nick (*nick*, *network*, *fn*)

Arguments

- **nick** (*string*) – The nickname you want to use now.
- **network** (*object*) – The network to execute the command on.
- **fn** (*function*) – The callback function to be called when the call has been finished.

Change your nickname to the specified nick.

2.1.4 channel api

class Channel (*name*, *client*, *network*)

Arguments

- **name** (*string*) – The channel name
- **client** (*object*) – Client object.

- **network** (*object*) – Network object.

Create a new channel object.

`Channel.`**`toString`**`()`

Returns string hostmask Hostmask of the channel.

`Channel.`**`getName`**`()`

Returns string Name Name of the channel.

`Channel.`**`getTopic`**`()`

Returns string Topic Topic of the channel.

`Channel.`**`getNames`**`()`

Returns object names Nicks in the channel: `{'nick': ['~']}`

`Channel.`**`getNetwork`**`()`

Returns string Network Network of the channel.

`Channel.`**`userHasMode`**`(`*user*, *mode*`)`

Arguments

- **user** (*object*) – The user to check the mode of in the channel.
- **mode** (*string*) – The mode to check for.

Returns boolean hasMode `true` if specified user has specified mode.

`Channel.`**`isUserInChannel`**`(`*user*`)`

Arguments

- **user** (*object*) – The user to check.

Returns boolean hasMode `true` if specified user is in this channel.

`Channel.`**`notice`**`(`*msg*`)`

Arguments

- **msg** (*string*) – Notice message to send to the user.

`Channel.`**`say`**`(`*msg*`)`

Arguments

- **msg** (*string*) – Message to send to the user.

`Channel.`**`reply`**`(`*user*, *msg*`)`

Arguments

- **user** (*object*) – User to reply to.
- **msg** (*string*) – Message to send to the user.

`Channel.`**`kick`**`(`*user*, *reason*`)`

Arguments

- **user** (*object*) – User to kick from channel.
- **reason** (*string*) – Reason for the kick.

`Channel.`**`ban`**`(`*mask*`)`

Arguments

- **mask** (*string*) – Hostmask to ban.

`Channel.`**`unban`**(*mask*)

Arguments

- **mask** (*string*) – Hostmask to unban.

events

`invite`

The `invite` event, fired when someone gets invited by someone.

Event attributes:

- `channel` - Channel you got invited to.
- `user` - User who sent the invite.
- `target` - Invited user.

Example:

```
client.on('invite', function (err, event) {
    console.log(event.target.getNick() + " got invited to "
    + event.channel.getName() + " by " + event.user.getNick());
});
```

`topic`

The `topic` event, fired when the topic gets changed. (or is originally sent)

Event attributes:

- `topic` - Current topic.
- `user` - User who changed the topic.
- `time` - Time of topic change.
- `changed` - `true` if topic was changed in this event.
- `channel` - Affected channel.
- `network` - Affected network.

Example:

```
client.on('topic', function (err, event) {
    console.log(event.channel.getName() + ":", event.topic);
});
```

`join`

The `join` event, fired when someone joins a channel.

Event attributes:

- `user` - User who joined.
- `channel` - Channel that was joined.

Example:

```
            client.on('join', function (err, event) {
                console.log(event.user.getNick() + " joined " + event.channel.getName());
});
```

names

The `names` event, fired when getting users in the channel.

Event attributes:

- `channel` - Affected channel.
- `names` - List of users in the channel.

Example:

```
            client.on('names', function (err, event) {
                console.log(event.channel.getName() + ":", event.names);
});
```

mode

The `mode` event, fired when a mode gets changed.

Event attributes:

- `mode` - Current mode.
- `channel` - Affected channel.
- `by` - User who changed the mode.
- `argument` - Mode argument.
- `adding` - boolean

Example:

```
            client.on('mode', function (err, event) {
                console.log(event.channel.getName() + ":", event.mode);
});
```

kick

The `kick` event, fired when a user gets kicked.

Event attributes:

- `channel` - Affected channel.
- `user` - Affected user.
- `by` - User who changed the kick.
- `reason` - Reason for the kick.

Example:

```
            client.on('kick', function (err, event) {
                console.log(event.channel.getName() + ":", event.user.getNick(), "was kicked.");
});
```

part

The `part` event, fired when a user parts a channel (channels).

Event attributes:

- `user` - Affected user.
- `channels` - Affected channels (channelList).
- `message` - Part message.

Example:

```
            client.on('part', function (err, event) {
                console.log(event.user.getNick(), "parted channels", event.channels);
});
```

functions

getChannelList()

Return array channelList List of channels.

Get a list of channels.

getChannel(*name*, *network*)

Arguments

- **name** (*object*) – The name of the channel you want to get.
- **network** (*object*) – The network to execute the command on.

Gets a channel by name.

isChannel(*channel*)

Arguments

- **channel** (*object*) – The channel object you want to check.

Checks if the passed object is a valid channel object.

invite(*name*, *channel*, *network*, *fn*)

Arguments

- **name** (*string*) – The name of the user you want to invite.
- **channel** (*object*) – The channel you want to invite him to.
- **network** (*object*) – The network to execute the command on.
- **fn** (*function*) – The callback function to be called when the call has been finished.

Invites a user to a channel.

topic(*channel*, *topic*, *network*, *fn*)

Arguments

- **channel** (*object*) – The channel you want to set the topic in.
- **topic** (*string*) – The topic you want to set.
- **network** (*object*) – The network to execute the command on.
- **fn** (*function*) – The callback function to be called when the call has been finished.

Sets the topic of a channel.

join(*channels*, *keys*, *network*, *fn*)

Arguments

- **channels** (*array*) – The channels you want to join.
- **keys** (*array*) – The keys for the channels you want to join.
- **network** (*object*) – The network to execute the command on.
- **fn** (*function*) – The callback function to be called when the call has been finished.

Joins channels.

ircNames (*channel*, *network*, *fn*)

Arguments

- **channels** (*array*) – The channel you want to get the nicknames from.
- **network** (*object*) – The network to execute the command on.
- **fn** (*function*) – The callback function to be called when the call has been finished.

Gets users from a channel.

ircMode (*target*, *flags*, *network*, *fn*)

Arguments

- **target** (*string*) – Target for the mode change, can be a user or channel.
- **flags** (*string*) – Flags of the mode change.
- **network** (*object*) – The network to execute the command on.
- **fn** (*function*) – The callback function to be called when the call has been finished.

Sets modes.

ircKick (*channels*, *nicks*, *msg*, *network*, *fn*)

Arguments

- **channels** (*array*) – The channels you want to kick from.
- **nicks** (*array*) – The nicks you want to kick.
- **network** (*object*) – The network to execute the command on.
- **fn** (*function*) – The callback function to be called when the call has been finished.

Kick user from a channel.

ircPart (*channels*, *msg*, *network*, *fn*)

Arguments

- **channels** (*array*) – The channels you want to kick from.
- **msg** (*string*) – The part message.
- **network** (*object*) – The network to execute the command on.
- **fn** (*function*) – The callback function to be called when the call has been finished.

Parts channels.

2.1.5 message plugins

events

message

The `message` event, fired when a standard IRC message is received.

Event attributes:

- `user` - User who sent the message.
- `channel` - Channel this message was sent to.
- `message` - The actual message.
- `isAction` - `true` if this was an action (/me).

Example:

```
// From README
client.on('message', function (err, event) {
  console.log('[' + event.channel.getName() + '] ' + event.user.getNick() + ': ' + event.message);
        event.reply('I logged to the console!'); // Says to the relevent user "I logged to the conso
});
```

privatemessage

The `privatemessage` event, fired when an IRC private message is received. Like a `message`, but more private.

Event attributes:

- `user` - User who sent the message.
- `message` - The actual private message.
- `isAction` - `true` if this was an action (/me).

Example:

```
// From README
client.on('privatemessage', function (err, event) {
  console.log('[PM] ' + event.user.getNick() + ': ' + event.message);
        event.reply(':)'); // Says to the relevent user ":)", in PM
});
```

notice

The `notice` event, fired when an IRC notice is received. Like a `message`, but more private.

Event attributes:

- `from` - User who sent the notice.
- `to` - Where this notice was sent to.
- `message` - The actual notice message.

Example:

```
            // From README
            client.on('notice', function (err, event) {
              console.log('[' + event.to + '] ' + event.from.getNick() + ': ' + event.message);
});
```

functions

send(*target*, *msg*, *network*, *fn*)

Arguments

- **target** (*object*) – The `channel` or `user` object to send this message to.
- **msg** (*string*) – The message you want to send.
- **network** (*object*) – The network to execute the command on.
- **fn** (*function*) – The callback function to be called when the call has been finished.

Send an IRC message to a channel or a user.

action(*target*, *msg*, *network*, *fn*)

Arguments

- **target** (*object*) – The `channel` or `user` object to send this action to.
- **msg** (*string*) – The action you want to send.
- **network** (*object*) – The network to execute the command on.
- **fn** (*function*) – The callback function to be called when the call has been finished.

Send an IRC action to a channel or a user. (This is the /me command)

notice(*target*, *msg*, *network*, *fn*)

Arguments

- **target** (*object*) – The `channel` or `user` object to send this notice to.
- **msg** (*string*) – The notice you want to send.
- **network** (*object*) – The network to execute the command on.
- **fn** (*function*) – The callback function to be called when the call has been finished.

Send an IRC notice to a channel or a user.

2.1.6 format plugin

formatting

The format plugin is a bit special. You can use various formattings by using the *client.format* object.

```
event.reply(client.format.red + 'Roses are red.' + client.format.blue + 'Violets are blue.' + client
```

The following colors and formatting options are available: https://github.com/caffeinery/coffea/blob/master/lib/plugins/format.js#L2-L23

emoji

We expose `require('node-emoji').emoji` via `client.emoji`. See the node-emoji documentation for more information: https://github.com/omnidan/node-emoji

kaomoji

We expose `require('node-kaomoji').kaomoji` via `client.kaomoji`. See the node-kaomoji documentation for more information: https://github.com/omnidan/node-kaomoji

functions

unhighlight (*message*)

Arguments

- **message** (*string*) – The message to be unhighlighted.

Returns string message Unhighlighted message (does not highlight users)

Adds ZWSP to a `message` to make sure it doesn't highlight anyone.

2.1.7 motd plugin

events

motd

The `motd` event, fired when the end of the MOTD (Message Of The Day) is received from the server.

Event attributes:

- `motd`: The actual MOTD sent by the server.

```
client.on('message', function (err, event) {
  console.log("Connected to " + event.network + ". MOTD: " + event.motd);
  client.join('##test'); // autojoins a channel when properly connected
});
```

2.1.8 welcome plugin

events

welcome

The `welcome` event, fired when the `RPL_WELCOME` command is received from the server. When this is fired `client.me.nick` gets set correctly and `client.welcomed` gets set to `true`.

Event attributes:

- `nick`: Your nick in the welcome message received from the server.
- `message`: The rest of the welcome message.

```
client.on('welcome', function (err, event) {
  console.log("Welcome " + event.nick + ": " + event.message);
});
```

2.1.9 server plugins

events

quit

The `quit` event, fired when the `QUIT` command is received from the server. When this is fired someone has quit from the irc network.

Event attributes:

- `user`: The affected user.
- `message`: The quit message.

```
client.on('quit', function (err, event) {
  console.log(event.user.getNick() + " quit: " + event.message);
});
```

functions

getServerInfo (*network*)

Arguments

- **network** (*object*) – The network to execute the command on.

Returns object serverInfo The server info about the network.

2.1.10 cap plugins

Warning: This part of the documentation is a work in progress and might be incomplete.

events

cap_list

cap_ack

cap_nck

Fired when `CAP` is received from the server. (More information available at https://github.com/ircv3/ircv3-specifications/blob/master/specification/capability-negotiation-3.1)

Event attributes:

- `capabilities`: The list of capabilities

extended-join

The `extended-join` event, fired when the client has the `extended-join` capability, and a user has joined a channel.

Event attributes:

- `channel`: The channel the user joined.
- `user`: The nick of the user.

- `account`: The host of the user.
- `realname`: The realname of the user.

2.1.11 away plugin

events

away

The `away` event, fired when an IRC AWAY event is received.

Event attributes:

- `user` - A User object of the user that changed away status.
- `message` - The message of the AWAY user's AWAY status (away message).

Example:

```
client.on('away', function (err, event) {
  console.log(event.user.getNick() + ' is now away: ' + event.message);
});
```

functions

away(*reason*, *network*, *fn*)

Arguments

- **reason** (*string*) – The reason to be away (away message).
- **network** (*object*) – The network to execute the command on.
- **fn** (*function*) – The callback function to be called when the call has been finished.

Sets the client as away on `network` with an away message (`reason`).

2.1.12 ping plugin

events

pong

The `pong` event, fired when a `PONG` event was received from the server.

Event attributes:

- None.

Example:

```
            client.on('pong', function (err, event) {
    console.log("PONG");
});
```

2.1.13 other plugins

events

errors

The `errors` event, fired when IRC errors are received. List of possible errors: https://github.com/williamwicks/irc-replies/blob/master/replies.json#L113-L170

Event attributes:

- None

Example:

```
client.on('errors', function (err, event) {
  console.error('IRC Error:', err);
});
```

functions

pass (*pass*, *network*, *fn*)

Arguments

- **pass** (*string*) – The password
- **network** (*object*) – The network to execute the command on.
- **fn** (*function*) – The callback function to be called when the call has been finished.

Sends an IRC PASS command to the network with the specified password (`pass`)

user (*username*, *realname*, *network*, *fn*)

Arguments

- **username** (*string*) – The username
- **realname** (*string*) – The realname
- **network** (*object*) – The network to execute the command on.
- **fn** (*function*) – The callback function to be called when the call has been finished.

Sends an IRC USER command to the network with the specified username (`username`) and realname (`realname`).

oper (*name*, *password*, *network*, *fn*)

Arguments

- **name** (*string*) – The oper name
- **password** (*string*) – The oper password
- **network** (*object*) – The network to execute the command on.
- **fn** (*function*) – The callback function to be called when the call has been finished.

Sends an IRC OPER command and tries to oper to the network with the specified name (`name`) and password (`password`).

Q

R

S

T

U

W